FedSTGNN: A Federated Spatio-Temporal Graph Neural Network

Heeyong Yoon
Department of Electrical Engineering and Computer Science
DGIST
Daegu, Republic of Korea
sunrise2575@dgist.ac.kr

Kang-Wook Chon*
School of Computer Science and Engineering
KOREATECH
Cheonan, Republic of Korea
kw.chon@koreatech.ac.kr

Min-Soo Kim*
School of Computing
KAIST
Daejeon, Republic of Korea
minsoo.k@kaist.ac.kr

***Abstract*—Owing to the explosive growth of the Internet of Things (IoT), there have been vast volumes of sensor-generated time series data in various locations. A lot of network usage occurs through these locations to process the sensor-generated data. To reduce network usage, the point of data processing gradually shifts from the central cloud to servers at the edge of the Internet, called edge servers. To cover the paradigm shift, there has been a challenging issue in training a neural network model, which aggregates the data generated by different locations and causes a massive amount of transmitted data across the network. Federated learning successfully resolves this issue by exchanging model parameters in the edge server through the central cloud. Meanwhile, Spatio-Temporal Graph Neural Networks (STGNNs) have gained much attention for analyzing time series data transmitted from several locations in IoT environments. Despite the increasing number of STGNN models that have been examined, the integration of STGNNs and federated learning remains underexplored. This paper presents FedSTGNN, a framework that seamlessly adapts arbitrary STGNN models to edge computing environments. The proposed method partitions the spatio-temporal graph with a mathematical definition and then trains each partitioned data with an arbitrary STGNN model on edge servers separately. Then, it aggregates all the model parameters in the central cloud with dramatically reduced network usage. Through experiments, we demonstrate that the FedSTGNN framework could train the model with a reduced amount of network communication by utilizing the advantages of edge computing and a slight loss in accuracy.**

***Index Terms*—spatio-temporal graph neural network, federated learning, IoT, machine learning framework, traffic data**

I. Introduction

The widespread use of mobile devices and the emergence of the Internet of Things (IoT) have resulted in the spread of numerous sensors and small-scale computer devices, which generate massive amounts of heterogeneous data. The flood of data has caused the problem of oversaturation of communication networks (called the *network traffic problem*), as all data is collected to centralized servers across the Internet network [1].

To handle the network traffic problem, *edge computing* [2] that moves computations from the central cloud to the edge servers of the Internet has been proposed. The edge computing could make the computations closer to the data sources. This strategy avoids network saturation because each edge device collects data individually in each region and then transmits only a small amount of data (i.e., the computation results instead of whole datasets). Fig. 1 shows an example of the difference between centralized computation and edge computing. In the central cloud-only method, the central cloud m receives all the information from all sensors, while in the edge computing environment, the edge server m_1 and m_2 receive local sensor information and transfer only essential information from raw sensor value to the central cloud m. Using this strategy, edge computing can reduce Internet traffic and offload computation to the edge of the Internet.

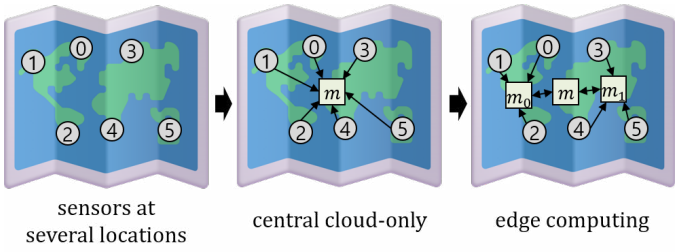

Fig. 1. Difference between centralized computation and edge computing. The circle with a number is a sensor (i.e., data source), m is the central cloud, m_0 and m_1 are edge servers.

As the data size from sensors and computing power from edge servers increase, a new approach, *federated learning* [3], has been proposed. This method is one of the distributed machine learning methods and exchanges only the model parameters between the edge servers and the central cloud. It moves the model training jobs to the edge servers closer to data sources (i.e., sensors). Specifically, each edge server trains its local model and then transmits each parameter to aggregate all the parameters trained in each edge server to the central cloud. Here, the central cloud updates the global model using the parameters from the local models.

By handling traffic, climate, or several time-based information from numerous sensors, *Spatio-Temporal Graph Neural Networks* (STGNN) [4]–[7] have recently gained popularity

* corresponding authors

to extract insights from raw sensor data. This model considers sensors scattered across the area as a spatio-temporal graph and applies neural network operation, including Graph Neural Network (GNN), to predict future sensor values or classify outliers. The spatio-temporal graph data is considered as a combination of spatial and temporal information: a weighted graph from sensor locations and multiple time series on every vertex from sensor output values. Fig. 2 depicts an example of spatio-temporal graph. Each sensor produces continuously measured values over time. In order to consider the values as the spatio-temporal graph, each sensor becomes a vertex, edges between sensors can be drawn based on the sensor proximity (i.e., the distance between sensors), and all the values produced from sensors are considered as time series data.

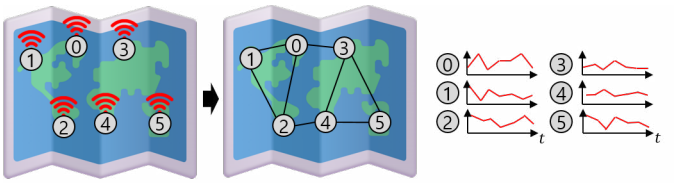

Fig. 2. Example of a spatio-temporal graph.

Since the spatio-temporal graphs grow along two axes (i.e., spatial scale and time scale), the required training time and amount of transmitted data via the network can be easily larger than a single-axis dataset like a set of images or words so that it can increase the network usage during training under typical centralized training methods. Therefore, applying federated learning to STGNN models has become more critical.

In this study, we propose a federated learning framework for STGNN, named **Fed**erated **S**patio-**T**emporal **G**raph **N**eural **N**etwork (FedSTGNN). This framework integrates the concepts of STGNNs and federated learning, which have not been presented. By utilizing the integrated concept, the proposed framework could reduce network utilization during training spatio-temporal graphs even though the data sizes are growing across edge devices. The proposed framework provides the generalized method, which could replace the arbitrary STGNN model with the function of federated learning, powered by the rigorous definition of federated learning for spatio-temporal graphs. Our framework shows acceptable accuracies (i.e., sensor value prediction) compared to the existing STGNN models that simply aggregate parameters.

To summarize, our main contributions are as follows:

- **Flexible framework**: We propose a general framework for training a spatio-temporal graph in federated learning, which does not depend on a specific model by its definition. To ensure that the model part can be replaced with an arbitrary model, we define local STGNN prediction using the concept of local vertex value.
- **Decent prediction and network cost performance**: This framework transforms an arbitrary centralized STGNN model into a federated version of the STGNN model. Despite learning with separated data on each edge server, it shows adequate prediction loss, which can make training harder than a centralized model. Also, edge computing empowers the framework to minimize network utilization between edge servers and the central cloud rather than directly sending information from sensors to the central cloud.

The rest of this paper is organized as follows. In Section II, we explain the essential concepts for understanding STGNN and federated learning. Section III explains the detail of our framework. We analyze and discuss the performance of our framework in Section IV. Finally, Section V concludes the paper.

II. Preliminaries

We describe how the sensor environment can be viewed as a spatio-temporal graph and its prediction method in Section II-A, and explain the operation of federated learning with some related works in Section II-B.

A. Spatio-temporal graph neural networks

A spatio-temporal graph is an abstract data type combining spatial and temporal data. Assume that sensors are installed in specified areas to collect data. We can consider each sensor as a vertex $v \in \mathcal{V}$ and the relationship between them (e.g., distance) as a weighted edge $(u, v, w) \in \mathcal{E}$, where u and v $(u, v \in V)$ are vertices, and w is the weight between u and v. Each sensor v collects values $x_v^{(t)} \in \mathcal{X}$ at each time $t \in \mathcal{T}$. As a result, spatial information, such as sensor locations, can be represented as a weighted graph, and temporal information, such as sensor value, can be represented as a time series. By combining the weighted graph and the time series, a spatio-temporal graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{T}, \mathcal{X})$ can be formulated.

For the sake of simplicity, we can combine $(\mathcal{V}, \mathcal{E})$ into adjacency matrix A and vertex value set $\mathcal{X}$ into vertex value matrix X. Finally, we can define a spatio-temporal graph using only matrices.

Definition 1 (Spatio-temporal graph)**.** A spatio-temporal graph $\mathcal{G}$ is defined as

$$\mathcal{G} = (A, X), \tag{1}$$

where $A \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ is an adjacency matrix, and $X \in \mathbb{R}^{|\mathcal{T}| \times |\mathcal{V}|}$ is a vertex value matrix. Note that $\mathcal{V}$ is a vertex set, $\mathcal{T}$ is a timestamp set, and $\mathbb{R}$ is the set of real numbers.

To simplify the representation of the part of the matrix in the following discussion, we now use the following notations for the matrix. For a given two number $0 \leq i < I$ and $0 \leq j < J$ for a matrix $B \in \mathbb{R}^{I \times J}$, an element at row i and column j can be represented as $B_{i,j}$. Using a colon (:), a matrix B can be sliced into smaller matrices. When the colon is alone, it means all the possible ranges of row or column of the matrix, therefore $B_{i,:} \in \mathbb{R}^J$. For x and y where it satisfies $x < y$, the notation $x{:}y$ represents a slice of the matrix following either row or column axis such as $B_{x:y,:} \in \mathbb{R}^{(y-x) \times J}$.

To insert the given spatio-temporal graph $\mathcal{G}$ as the input of an STGNN model, $\mathcal{G}$ needs to be preprocessed using a

sliding window [8]. This data extraction method generates multiple short time series by overlapping a fixed-size window whose length of temporal axis is $t_{\text{in}}+t_{\text{out}}$. The sliding window generates the larger training datasets than those of the original time length $|\mathcal{T}|$. More specifically, for all timestamp $t \in \mathcal{T}$ that satisfies $t_{\text{in}} \leq t \leq |\mathcal{T}| - t_{\text{out}}$, the sliding window method extracts all possible slices of matrix $X_{t-t_{\text{in}}:t+t_{\text{out}},:} \in \mathbb{R}^{(t_{\text{in}}+t_{\text{out}})\times|\mathcal{V}|}$ from X. Each slice of matrix is divided into two part; $X_{t-t_{\text{in}}:t,:} \in \mathbb{R}^{t_{\text{in}}\times|\mathcal{V}|}$ and $X_{t:t+t_{\text{out}},:} \in \mathbb{R}^{t_{\text{out}}\times|\mathcal{V}|}$. Then we consider $X_{t-t_{\text{in}}:t,:}$ as the model input and $X_{t:t+t_{\text{out}},:}$ as the model output to train the model f_θ by the following Eq. (2).

Definition 2 (STGNN prediction)**.** For each $t \in \mathcal{T}^*$, an STGNN prediction model f_θ can predict t_{out} time length of future vertex value $\hat{X}_{t:t+t_{\text{out}},:}$ from t_{in} time length of past vertex value $X_{t-t_{\text{in}}:t}$, with the adjacency matrix A, such as

$$\hat{X}_{t:t+t_{\text{out}},:} = f_\theta(A, X_{t-t_{\text{in}}:t,:}), \tag{2}$$

where $\mathcal{T}^* = \{t \mid t \in \mathcal{T}, t_{\text{in}} \leq t \leq |\mathcal{T}| - t_{\text{out}}\}$. The predicted future value $\hat{X}_{t:t+t_{\text{out}},:}$ is then compared to the actual future value $X_{t:t+t_{\text{out}},:}$ for calculating the mean prediction loss ℓ using a loss function $\mathcal{L}$ as

$$\ell = \frac{1}{|\mathcal{T}^*|}\sum_{t\in\mathcal{T}^*} \mathcal{L}(X_{t:t+t_{\text{out}},:}, \hat{X}_{t:t+t_{\text{out}},:}). \tag{3}$$

B. Federated learning

Federated learning is first shown in the study [3]. This learning method is achieved by repeatedly executing several *round*s: each edge server m_i receives local training data from its data source and trains separately by a few epochs η. The edge server m_i sends only its local model parameter θ_i to the central cloud m. The central cloud m updates the global model parameter θ by averaging the received local parameters. The updated global parameter θ is replicated to the local model parameter θ_i. The global and local model parameters are updated, and eventually, it converges model loss by repeating ρ rounds.

Some studies have tackled the idea that edge servers can contribute differently during training. AgnosticFL [9] raises the issue that not all edge servers contribute to learning models equally. Unlike the original federated learning method, which accepts the local parameters fairly, this study applies a contribution weight λ for the global model on each edge server and updates all λ weights. FedOV [10] resolves the overconfidence classification problem induced by each edge server's biased local data. Specifically, this method suggests that edge servers vote to postpone the classification of the out-of-distribution data.

Another type of federated learning enhancement focuses on reducing total learning times. The computation power of each edge server can vary widely, such as the difference between a low electric power device and a high-performance GPU server, which can lead to delayed training time bounding to the slowest edge server. FedProx [11] proposes a method adjusting the number of local epochs per round and per edge server. SplitFed [12] shows a different approach that partitions the whole model into sub-models in order to train models cooperatively on both the central cloud and edge server, rather than offloading the entire model training process on the edge server side.

III. FEDSTGNN

This section introduces our framework, FedSTGNN, and its detailed workflows. Before explaining the overall framework, we explain the local vertex value $X^{(i)}$, local prediction method, and global aggregation process.

In the edge computing environment, it is preferred that each edge server receives information from nearby sensors to reduce network usage. Therefore, each edge server only sees a partial part of the data during the federated learning. In the spatio-temporal graph, each sensor is considered as a vertex $v \in \mathcal{V}$, so we can define the local vertex value $X^{(i)}$, which is the partial part of the X.

Definition 3 (Local vertex value)**.** The vertex is partitioned into several disjoint sets $\mathcal{V}^{(i)} \subseteq \mathcal{V}$ for each i-th edge server. In this case, each edge server only processes partial vertex value $X^{(i)} \in \mathbb{R}^{|\mathcal{T}|\times|\mathcal{V}|}$, where all the matrix elements corresponding to absent vertices are zero-filled, which is, for every $v \in \mathcal{V}$,

$$X^{(i)}_{:,v} = \begin{cases} X_{:,v} & v \in \mathcal{V}^{(i)} \\ 0 & \text{otherwise} \end{cases}. \tag{4}$$

In the same manner of Definition 2, the local vertex value $X^{(i)}$ from Eq. (4) is divided into several slices $X^{(i)}_{t-t_{\text{in}}:t+t_{\text{out}},:} \in \mathbb{R}^{(t_{\text{in}}+t_{\text{out}})\times|\mathcal{V}|}$ such that $t \in \mathcal{T}^*$, and partitioned into input and output parts, $X^{(i)}_{t-t_{\text{in}}:t,:}$ and $X^{(i)}_{t:t+t_{\text{out}},:}$, respectively. Because the location of each sensor does not rapidly change through time like the traffic or climate sensors, all edge servers can be aware of the location of each sensor; therefore, adjacency matrix A in Eq. (1) commonly exists for all edge servers. By the above concept, we can define federated STGNN prediction.

Definition 4 (Local STGNN prediction)**.** Each i-th edge server predicts the local predicted value $\hat{X}^{(i)}_{t:t+t_{\text{out}},:}$ by the following equations.

$$\hat{X}^{(i)}_{t:t+t_{\text{out}},:} = f_{\theta_i}(A, X^{(i)}_{t-t_{\text{in}}:t,:}), \tag{5}$$

such that $t \in \mathcal{T}^*$ from Definition 2, and f_{θ_i} is a local model on each i-th edge server containing its local model parameter θ_i. The loss of the model is computed by the same equation of Eq. (3), only replacing all X in the equation to $X^{(i)}$.

Using Eq. (5), each local model f_{θ_i} is trained. These trained local models are then aggregated into a global model in the central cloud. We apply FedAvg [3] to the aggregation, averaging each local model parameter to the global model parameter and broadcasting them back to the edge servers.

Fig. 3 shows the overall process of FedSTGNN. The local vertex value $X^{(i)}$ is transmitted into i-th edge server, and the model f_{θ_i} in each edge server receives local sensor data with pre-given adjacency matrix A at step ①. Each edge server independently trains and updates its local model parameter θ_i in step ②. In step ③, each edge server uploads its

model parameter to the central cloud through networks. After uploading model parameters, the central cloud averages all received local parameters θ_i and applies them into the global model parameter θ. The integrated model is then distributed back to the edge server and is overwritten to each local model at step ④. By repeatedly training local and global models through step ① to ④, all parameters in each edge server are gradually trained without exchanging data between edge servers. Note that the process from step ① to ④ is called a *round*; therefore, if there are ten rounds, the central cloud parameter θ is updated ten times.

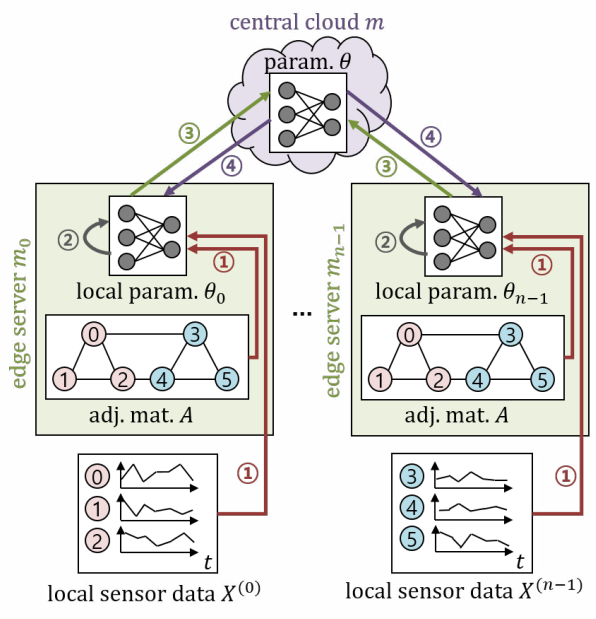

Fig. 3. Overview of FedSTGNN.

The mechanism of FedSTGNN can be represented as Algorithm 1. At first, the framework prepares the loss record list $\mathcal{R}$ for the performance evaluation (Line 1 of Algorithm 1). By each round r, FedSTGNN performs the same procedure for every i-th edge server (Lines 2–3 of Algorithm 1). The model f_{θ_i} is updated by several arguments as mentioned in Eq. (5), and it validates model performance to get the mean local model loss ℓ_i (Line 4 of Algorithm 1). Each model parameter is collected and averaged in the central cloud to overwrite the global model parameter θ (Line 5 of Algorithm 1). The global model f_θ is then tested for the framework evaluation (Line 6 of Algorithm 1). Finally, the central cloud broadcasts global model parameter θ to each local model parameter θ_i (Lines 7–8 of Algorithm 1). The tuple of global mean loss ℓ and local mean loss ℓ_i is appended to the loss record $\mathcal{R}$ for the performance examination (Lines 9–10 of Algorithm 1).

The structure of FedSTGNN leads to various advantages. First, since only model parameters θ_i are transferred between the edge server and the central cloud, FedSTGNN can reduce significant network utilization compared to the central cloud-only case. In central cloud-only case, every j-th sensor in $\mathcal{V}$ sends value $X_{t,j}$ such that $t \in \mathcal{T}$, so the total network usage

Algorithm 1 FedSTGNN

Data: A, /* Adjacency matrix of sensors */
$X_i \subseteq X$, /* i-th partition of sensor data */
θ, /* a model parameter in the central cloud */
$\theta_i \in \Theta$, /* a model parameter in i-th edge server */
ℓ, /* mean loss of the central cloud model */
ℓ_i, /* mean loss of i-th edge server model */
ρ, /* the number of rounds */
η, /* the number of epochs */
M, /* a set of i-th edge server m_i */

1: $\mathcal{R} \leftarrow []$ /* loss records of each round */
2: **for each** $r \in [0, ..., \rho - 1]$ **do**
3: **for each** $i \in [0, ..., |M| - 1]$ **do**
4: $\theta_i, \ell_i \leftarrow$ TrainingAndTestModel(θ_i, A, X_i, η)
5: $\theta \leftarrow \frac{1}{|M|} \sum_{i=0}^{|\Theta|-1} \theta_i$
6: $\ell \leftarrow$ TestModel(A, X) /* for the evaluation */
7: **for each** $i \in [0, ..., |M| - 1]$ **do**
8: $\theta_i \leftarrow \theta$
9: $\mathcal{R} \leftarrow \mathcal{R} \cup (\ell, \ell_0, ..., \ell_{|M|-1})$
10: **return** $\mathcal{R}$

is $O(|\mathcal{T}| \times |\mathcal{V}|)$. In the FedSTGNN case, per each round ρ, each edge server $m_i \in M$ sends its local parameter θ_i to the central cloud and then receives the updated parameter θ from the central cloud, so the total network usage is $O(2 \times |\theta_i| \times |M| \times \rho)$ such that $|\theta| = |\theta_i|$. In other words, the network usage of the central cloud-only case depends on the data (i.e., $\mathcal{T}$ and $\mathcal{V}$), while the FedSTGNN's network usage depends on the model and the setting of edge computing (i.e., $\mathcal{M}$, θ_i, ρ).

FedSTGNN's effect on network usage reduction is validated in a real-world scenario. For the central cloud-only case, the transmitted data size is calculated from the dataset METR-LA [4], which is $|\mathcal{V}| = 207$, $|\mathcal{T}| = 34272$, and each $X_{t,j}$ is 4-byte floating point value; therefore, the total transmitted size is $|\mathcal{V}| \times |\mathcal{T}| \times |X_{t,j}| \approx 27$MB. On the other hand, for the FedSTGNN case, the transmitted data size is calculated from the size of model parameters, which is ASTGCN [6] is $|\theta_i| \approx 50$KB, and edge computing settings, which is $|M| = 4$ and $\rho = 20$; therefore the total transmitted size is $2 \times |\theta_i| \times |M| \times \rho \approx 8$MB, which indicates that FedSTGNN can reduce network usage compared with those on the central cloud-only environment.

Additionally, the proposed framework does not depend on a specific STGNN model as defined in Definition 4. Here, the model f_{θ_i} could be replaced with arbitrary ones as presented in Line 4 of Algorithm 1. This feature could allow models to be readily tuned and replaced to improve accuracy with reduced training times.

IV. Evaluation

In this section, we show our experimental results on the FedSTGNN framework. We first introduce the experimental environment, including the dataset, model, and measure. Then,

we show the results of training models varying the use of the proposed framework. Finally, we analyze the network usage in the training process.

A. Environment

All experiments are run on a multi-node GPU cluster consisting of one central cloud node and four single GPU edge server nodes. All nodes have two Intel Xeon E5-2630v4 (2.20 GHz, each with ten cores and 20 threads), 512 GB of main memory, 512 GB of SSD, and one NVIDIA GTX 1080Ti. A 1 Gbps Ethernet switch connects all nodes.

For the model f_θ described in Eq. (5), we use a spatio-temporal graph neural network named ASTGCN [6], which repetitively applying of a spatio-temporal neural network block, consisting of attention and convolutional networks to handle spatial and temporal information. We use 64 graph and temporal convolutional kernels, which are the same as the default settings described in the paper [6].

We select two public datasets of the spatio-temporal graph $\mathcal{G}$ described in Eq. (1); **METR-LA** and **PEMS-BAY** [4], collected by the California Department of Transportation. Vertices in the dataset are vehicle detectors, and edges are the distance between vehicle detectors. Each sensor measures the average speed of passing vehicles over 5 min. METR-LA has 207 sensors and 34,272 timepoints, while PEMS-BAY has 325 sensors and 52,116 timepoints.

We partition all vertices $\mathcal{V}$ of these datasets into $\mathcal{V}^{(i)}$ from Definition 3, from no partition ($|M| = 1$) to four partitions ($|M| = 4$) using Spectral Clustering as depicted in Fig. 4. Then, we separate each sensor data X into $X^{(i)}$ following the partition of the vertices as shown in Eq. (4). As a result, each edge server has only A and $X^{(i)}$.

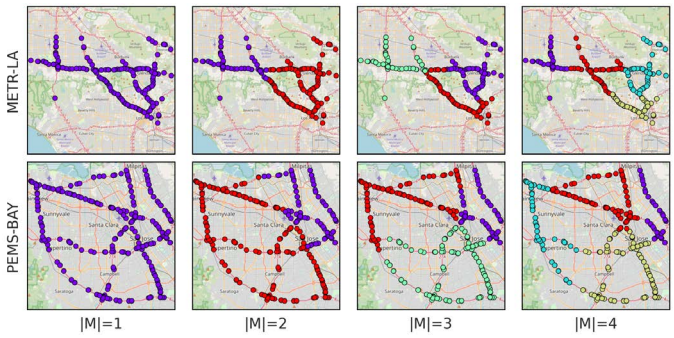

Fig. 4. Results of sensor partitioning. The vertical axis of all images is the name of the dataset, and the horizontal axis is the number of edge servers represented as $|M|$.

B. Effect of federated learning

We compare the model's prediction performance between FedSTGNN-applied and central cloud-only conditions. More specifically, we conduct our experiment by increasing the number of edge servers from one to four. The case with one client represents a central cloud-only environment; we use this value as a baseline. Otherwise, we use edge servers for two to four clients and compare them to the baseline. In all experiments, the number of rounds was set to $\rho = 20$, and the number of epochs was set to $\eta = 5$. We measure the performance as Mean Arctangent Absolute Percentage Error (MAAPE) [13], which calculates the error of predicted value relative to the actual value in the percentage. MAAPE is considered as an improved metric than Mean Absolute Percentage Error (MAPE), which is the commonly used metric, because it resolves a critical problem happening in MAPE; if the difference between the actual and predicted value is enormous, MAPE diverges to the infinite, while MAAPE converges to near 100%.

Fig. 5 shows the difference in loss depending on the number of clients. The experimental results show that the loss increases by about 5.2% in METR-LA and 1.8% in PEMS-BAY for the FedSTGNN case ($|M| = 2, 3, 4$), respectively, compared to the cloud-only case ($|M| = 1$). In the FedSTGNN condition, the loss increases slightly as the number of clients is increased. Even when the number of clients is four, the loss is at most 12.8% in METR-LA and 4.9% in PEMS-BAY, which means that FedSTGNN has a prediction accuracy of about 87.2% and 95.1%, respectively.

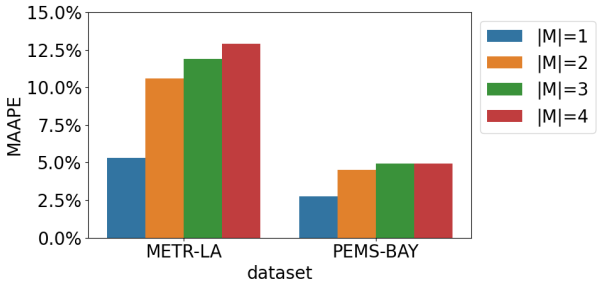

Fig. 5. MAAPE loss comparison by the number of clients. $|M|$ is the number of edge servers.

We examine the training process by round. Fig. 6 shows the validation loss change as the number of rounds increases during training. In both the central cloud case ($|M| = 1$) and the edge server cases ($|M| = 2, 3, 4$), MAAPE loss converges as the round increases. However, when using the central cloud, the loss converges relatively stable, while the edge server converges with more fluctuations.

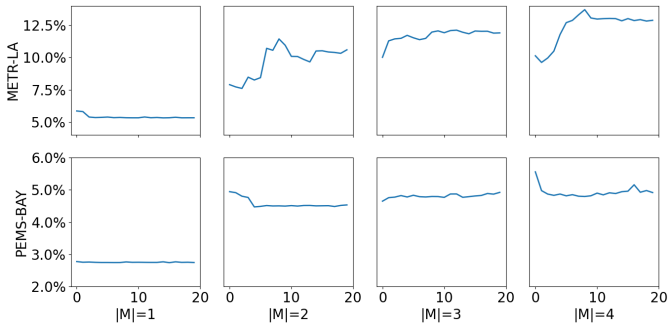

Fig. 6. Loss tendency by epochs. $|M|$ is the number of edge servers. The x-axis of each plot is round, and the y-axis is MAAPE.

C. Effect on network usage

We examine the network usage of our framework. We compare two scenarios: the case using central cloud-only and the case using FedSTGNN. To measure their impact on the global network, such as the Internet, we evaluate communication volume using different criteria for central cloud-only and FedSTGNN. This distinction is required because exchanging information between devices is different in the central cloud-only and edge computing environments, as illustrated in Fig. 1. We use different criteria in both scenarios to assess meaningful effects on network congestion. For the central cloud-only case, we measure network usage between sensors and the central cloud because sensors directly transmit information. In contrast, for the edge computing case, we measure network usage between edge servers and the central cloud. In this case, sensor data is initially sent to the nearest edge server (which has negligible impact on the global network), and the processed data is then delivered from the edge server to the central cloud via the global network.

Table I presents the average network usage reduction in training for each round with two datasets. The proposed framework FedSTGNN reduces a significant network traffic (i.e., 73% and 76% for the METR-LA and PEMS-BAY datasets, respectively). This reduction is induced by FedSTGNN's approach of transmitting only the local model parameters from edge servers to the central cloud. In contrast, the cloud-only scenario results in excessive network traffic, because STGNN processes all slices of X generated at each time $t \in \mathcal{T}^*$ as depicted in Definition 2 and iterates the same dataset for several epochs η.

TABLE I
AVERAGE NETWORK USAGE REDUCTION IN TRAINING PER ROUND.

Dataset	Central cloud-only	FedSTGNN ($\|M\| = 4$)	Reduction rate
METR-LA	27.06 MB	7.24 MB	73.3%
PEMS-BAY	64.61 MB	15.16 MB	76.5%

V. CONCLUSION

We propose FedSTGNN, a federated learning framework for spatio-temporal graph neural networks. We define the concept of STGNN based on matrices and extend it to support federated learning, enabling federated learning to arbitrary STGNN models. We employ a real-world traffic sensor dataset and an existing STGNN model for the evaluation and divide the sensors into adjacent groups to construct the experimental environment of edge computing. Through the experimental evaluation, we demonstrate that FedSTGNN shows only a modest performance degradation of 1.8% to 5.2% compared to the central cloud version model, which can be considered to preserve the original performance. Furthermore, FedSTGNN exhibits stable loss traces that converge to specific loss values as the synchronization rounds progress. Notably, our framework significantly reduces overall network traffic usage by up to 76.5%, and it demonstrates its suitability for edge computing environments.

ACKNOWLEDGMENT

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2024-2020-0-01795) supervised by the IITP(Institute of Information & Communications Technology Planning & Evaluation).

REFERENCES

[1] H. Chang, A. Hari, S. Mukherjee, and T. Lakshman, "Bringing the cloud to the edge," in *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 346–351, IEEE, 2014.

[2] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[3] L. Liu, J. Zhang, S. Song, and K. B. Letaief, "Client-edge-cloud hierarchical federated learning," in *ICC 2020-2020 IEEE international conference on communications (ICC)*, pp. 1–6, IEEE, 2020.

[4] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," 2018.

[5] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pp. 3634–3640, International Joint Conferences on Artificial Intelligence Organization, 7 2018.

[6] S. Guo, Y. Lin, N. Feng, C. Song, and H. Wan, "Attention based spatial-temporal graph convolutional networks for traffic flow forecasting," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, pp. 922–929, 2019.

[7] Z. Wu, S. Pan, G. Long, J. Jiang, and C. Zhang, "Graph wavenet for deep spatial-temporal graph modeling," *arXiv preprint arXiv:1906.00121*, 2019.

[8] L. Hällman, "The rolling window method: Precisions of financial forecasting," *Dissertation*, 2017.

[9] M. Mohri, G. Sivek, and A. T. Suresh, "Agnostic federated learning," in *International Conference on Machine Learning*, pp. 4615–4625, PMLR, 2019.

[10] Y. Diao, Q. Li, and B. He, "Towards addressing label skews in one-shot federated learning," in *The Eleventh International Conference on Learning Representations*, 2022.

[11] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *Proceedings of Machine learning and systems*, vol. 2, pp. 429–450, 2020.

[12] C. Thapa, P. C. M. Arachchige, S. Camtepe, and L. Sun, "Splitfed: When federated learning meets split learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, pp. 8485–8493, 2022.

[13] S. Kim and H. Kim, "A new metric of absolute percentage error for intermittent demand forecasts," *International Journal of Forecasting*, vol. 32, no. 3, pp. 669–679, 2016.